Department of Computer Science
George Mason University Technical Reports

4400 University Drive MS#4A5
Fairfax, VA 22030-4444 USA
http://cs.gmu.edu/ 703-993-1530

Adaptive Search for Interesting Things

William Squires wsquires@masonlive.gmu.edu
Sean Luke sean@cs.gmu.edu

Technical Report GMU-CS-TR-2012-4

Abstract

The results of a parameter sweep over a multidimensional parameter space are often used to gain an understanding of the space beyond simply identifying optima. But sweeps are costly, and so it is highly desirable to adaptively sample the space in such a way as to concentrate precious samples on the more "interesting" areas of the space. In this paper we analyze and expand on a previous work which defined such areas as those in which the slope of some quality function was high. We develop a performance metric in terms of generalizability of the resulting sampled model, examine the existing method in terms of scalability and accuracy, and then propose and examine two population-based approaches which address some of its shortcomings.

1 Introduction

There are two common reasons why one would iteratively, and adaptively, probe for the quality or nature of points in a space with regard to some function f. One is of course to find points which *optimize* that function in some sense (such as maximization). But another equally common reason is to collect a set of points which best help us understand the shape of the surface of f, perhaps to ultimately build up a model of the surface. This is usually known as *parameter sweeping*. For example, scientists often use parameter sweeps to understand the sensitivity of a surface to changes in various parameters.

One approach to doing this is to perform independent sensitivity analyses on each parameter of interest. The problem with this approach is that by projecting out the other parameters during a given sensitivity test, we completely disregard linkages and relationships among the parameters. Thus it is desirable to probe in the joint parameter space. However, depending on the complexity of the surface, the number of points required to accurately model the surface can become prohibitively large, and particularly so as the dimensionality of the joint parameter space grows. But points are expensive: each may represent an experimental test using those parameters. Thus the goal is to probe adaptively so as to maximize understanding of the nature of the surface with a minimum of samples.

To attack this issue Luke et al [8] proposed a model-free adaptive sampling approach based loosely on a population-based technique similar to an evolutionary algorithm. Their technique was to initially sample randomly, then begin selecting pairs of points which were likely to have some *interesting* but so-far unexplored feature of the space between them. Between each such pair Luke et al generated one or more children through an iterated bracketing technique. The goal was to sample points more densely on "interesting" regions of the space than on non-interesting regions.

Luke et al defined *interesting* regions as those for which the magnitude of the gradient of the surface was greatest — that is, where the surface had its steeper slopes. This was a reasonable assumption: to understand a surface from a set of samples, it was best to place samples on the steeper slopes. For example, if a surface had a broad swath of flat area, there was no need to waste samples in that region: just sample once and declare that everything else in the neighborhood resembled it.

However there were three major issues left open by this research. First, the effectiveness of the algorithm was measured by plotting the concentration of individuals in each range of "interestingness" (slope) in the surface, and arguing visually for the apparent correlation. Second, the bracketing approach produced individuals which were excessively crowded along the slope, reducing the effectiveness of the algorithm at minimizing the number of necessary samples. Third, the method was only tested for two dimensions of parameters: it was not clear if it would scale. Indeed, we suspected that the so-called *curse of dimensionality* was likely to rear its head.

This paper seeks to tackle these three issues:

1. We propose a significantly better measure of "interestingness". This boils down, fundamentally, to *generalizability* of the resulting sampled model. While this new definition allows us to perform quantitative assessment, it does not discard the philosophical underpinnings of the original approach: indeed it is strongly associated with it.
2. We improve the algorithm through adaptive bracketing and crowding in order to reduce undue sample density along the gradient.
3. We examine the scalability both of the original algorithm and the revised one.

1.1 Finding Interesting Things

In [8] a novel steady-state population based parameter sweeping algorithm was defined and demonstrated using three 2-dimensional problems. Although their algorithm was not an optimizer, they used EA terminology and so the surface function of interest f was effectively a fitness function. The algorithm, which we call the *Fixed Bracketing Sweep*, or FBS, is roughly defined as follows:

1. Generate an initial population of individuals over the parameter space using a uniform random distribution.
2. Choose parent 1 randomly from the population, or with some exploration probability (set to 0.1), introduce a new random sample as parent 1.
3. Choose parent 2 from the population using a double tournament (described below).
4. Repeat N times:
   (a) Generate a child somewhere along the line segment joining parent 1 and parent 2.
   (b) Among the child and two parents, select the two new "parents" 1 and 2 for the next iteration of the loop (described below).
5. If the maximum population size has not yet been reached, go to step 2.

Note that, unlike traditional population-based methods, this algorithm never removes individuals from its population: the population continues to grow and grow.

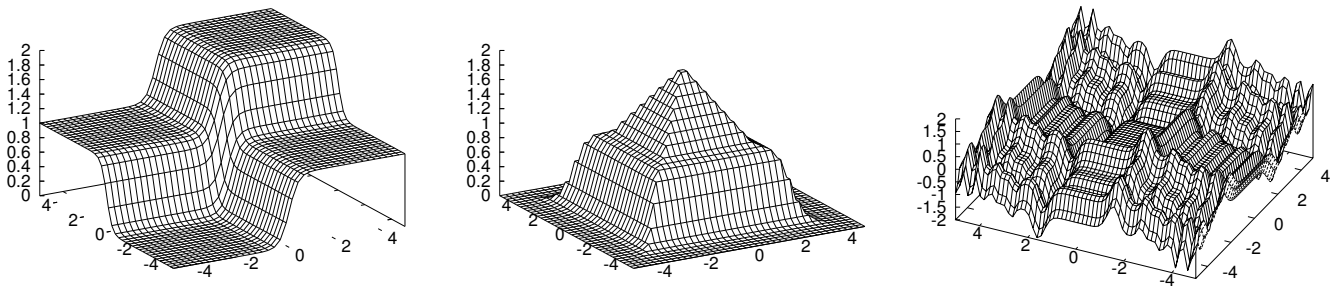
The Double Tournament Parent 2 is selected using a tournament of size 15, where the superior individual is one whose fitness is *most different* from Parent 1. Entrants to this tournament were not selected at random with replacement, but rather were the winners of 15 *other tournaments* at a second level. The second-level tournaments, each of size 10, treated the superior individual as the one who was *nearest* in distance to Parent 2. Entrants to these tournaments were selected at random with replacement. The idea behind this approach was to find parent pairs which were nearby but very different in f, thus making it more likely that between them the surface had a steep gradient somewhere.

Iterated Bracketing In the bracketing loop, the parents for the next bracketing iteration are the child and the parent that is *most different* from the child both in terms of distance and fitness. Specifically, the parent which minimized the function $\frac{||\text{location}(parent)-\text{location}(child)||}{|\text{fitness}(parent)-\text{fitness}(child|}$. Using this approach, children are successively created in the space with the likely steepest slope.

2 Related Work

Experimental Design Part of experimental design concerns itself with organizing a finite and costly set of samples (experiments) so as to maximize understanding of the underlying response surface of the environment [4]. In most cases a full sweep of all combinations of parameters is infeasible, and so we must content ourselves with some limited subsampling, either via deterministic but *fractional sampling*, which simply reduces the resolution of the parameter settings of interest, or through uniform random sampling. In some cases, where each parameter is discrete or discretizable, some strategies determine those points on a grid which, if projected in each dimension, maximize the number of different settings attempted for a given parameter. Perhaps the most well-known such approach is the *latin hypercube* [14]. Many techniques then fit a curve to the provided samples in order to simplify, interpolate, or extrapolate the results. Such curve fitting is generally straightforward regression, using models as simple as linear regression to neural networks or mixtures of gaussians: and this area has a huge literature (see [1] for a current overview of the myriad options). In the evolutionary computation realm, fitting curves to response surfaces has long been a goal of symbolic regression (for example, [15, 7, 11, 13]).

K-Nearest Neighbor In this paper we are focusing on the first step (sampling), rather than on the later step (modeling). However in order to assess the quality of our sampling method, we must fit a model to a set of validation points. For that, we apply the *3-Nearest Neighbor* algorithm. The general *K-Nearest Neighbor* (or K-NN) algorithm is among the simplest, and often most robust, predictors: any given point in space is modeled as the average, or voting result, among the three points in the original training sample which were closest to it. Traditionally K-NN has been used for binary classification (see again [1]), but variants of it may be used for curve-fitting. Typically a point is estimated as the average of its



Cross **Pyramid** **Michalewicz**

Figure 1: The Cross, Pyramid, and Michalewicz functions in 2D

three nearest neighbors from the original training sample, weighted in some way according to their distance from the point. In our work, we chose (for point $\vec{x}$ and nearest neighbor samples $y^{\vec{(1)}}, y^{\vec{(2)}}, y^{\vec{(3)}}$):

$$f(\vec{x}) = \frac{\sum_i w_i f(\vec{y}^{(i)})}{\sum_i w_i}$$

Where the weighting function is defined as:

$$w_i = \frac{1}{||\vec{x} - \vec{y}^{(i)}||^2}$$

In [6] it was proven that with sufficient training samples an accurate prediction of actual values within some ϵ is possible for continuous functions with bounded slope. Further it was proven that the number of samples required is polynomial in the dimensionality of the problem. The optimal value of K may vary from problem to problem [12], but $K = 3$ has long been the most popular choice and the default for many studies (including ours).

Crowding Crowding is a niching technique whereby individuals are chosen to die, or refused entry into the population, if their genotypical or phenotypical region contains too many individuals. Crowding is largely used as a diversity maintenance mechanism in evolutionary computation. In the original formulation of crowding [2], new children were introduced to a steady-state population by selecting (via tournament selection), killing off, and replacing individuals which were near to them in the space. Later versions [5] had the introduced child competing with the candidate selected to die. [9] addresses the problem of stochastic errors in replacement, which can result in eventual convergence to a single value in the population. A number of crowding and replacement algorithms have been developed since, such as Deterministic Crowding, Probabilistic Crowding, and other local tournament algorithms which use additional information in the replacement process [10]. Crowding also takes a central role in guaranteeing good diversity along Pareto fronts in modern multiobjective optimization algorithms, such as NSGA-II [3] and SPEA2 [16].

In this paper, crowding is used to control the parent selection process and avoid creating children in already crowded areas of parameter space. However, in our approach, crowding simply determines whether bracketing should continue adding children (that is, whether the loop in Line 4 of Section 1.1 should be cut off early). Thus in our form of crowding, children are simply prevented from being added to the population, and so we have no need for a notion of replacement.

3 Experimental Methodology

We propose a new approach for testing how well an algorithm samples the "interesting parts" of the environment: simply how well the resulting samples modeled a set of random and unseen validation points. Our procedure is as follows. We first run the algorithm in question and produce a sample (population) of points. We then feed these points into a 3-Nearest Neighbor learner, as described in Section 2, to build a model. Then we provide the model with randomly generated validation samples equal to the population size. The performance of the algorithm is the root mean square error over all the validation samples.

To see how this tests the "interesting" areas, consider the following. If too many points are wasted on open, flat regions of the space, then fewer points will lie on the areas of change and the total error will increase. The method has an advantage as well: because the 3-NN learner in question is an interpolation procedure, excessive points on unchanging slopes (flat tilted regions) will also not benefit much from excessive points.

We tested with three test problems: simple (*Cross*), moderate (*Step Pyramid*), and complex (*Michalewicz*). To examine scaling, we also tested with parameter dimensions 2 through 5 (only 2 was used in the original paper).

Each test was examined at ten equal population size intervals to assess algorithm performance as the population size increased. Tests were repeated 50 times for statistical accuracy, and confidence intervals were generated at each size interval. All of the plots in this paper include 95% confidence intervals, but the error is *very* small due to the large number of samples at the end of each run.

The test problems are all generalizable to any N dimensions, but the Figure 1 shows the two-dimensional cases. All functions were plotted in a space ranging from [-5, 5] in each dimension.

Cross The Cross function was selected from [8], and represents a relatively "easy" problem. We found the other two problems used in [8] to be similarly "easy" and so were omitted here. The cross function generalized to n dimensions is:

$$cross(\vec{x}) = \sum_{i=1}^{n} \sigma(x_i, 5)$$

where $\sigma(u, \beta)$ is the *sigmoid function*:

$$\sigma(u, \beta) = \frac{1}{1 + e^{-\beta u}}$$

Step Pyramid The Step Pyramid function is a medium level of complexity with less flat area than Cross. This function produces a ziggurat-like pyramid with various steps, controlled by the *step size* s, set to 2.0 for these experiments. Like Cross, the Step Pyramid function uses the sigmoid function to provide a continuous surface. The function is:

$$f(\vec{x}) = \frac{1}{1 + \sigma(-5s(\text{edgedist}(\vec{x}) - \text{step}(\vec{x})), 5)} + \text{step}(\vec{x}) - 1$$

where step and edgedist are defined as:

$$\text{step}(\vec{x}) = \max\left(1, \frac{\text{edgedist}(\vec{x}) - 1}{s}\right)$$
$$\text{edgedist}(\vec{x}) = \min_i(\min(5 - x_i, 5 + x_i))$$

Michalewicz The Michalewicz function is a well known test function for evolutionary algorithms which produces a complex surface with many peaks and valleys. The function was chosen as a "difficult" function for the algorithm, as it had very few flat regions. The value for m controls the steepness of the peaks and valleys in the function. We set $m = 2$:

$$f(\vec{x}) = -\sum_{i=1}^{n} \sin(x_i) \sin(x_i^2 i/\pi))^{2m}$$

We do not show results for the Michalewicz function because we found that all tested methods performed universally poorly on it: it appears that this technique is only effective for surfaces with some degree of "uninterestingness" to them to allow adaptive sampling.

4 First Experiment: Fixed Bracketing

We began by analyzing the original algorithm in [8] under different bracket sizes, and comparing them to uniformly randomly distributed points. In [8] both bracket sizes of 1 and 5 were shown, though only a bracket size of 1 was tested. We wished to understand the effect of increasing bracketing had on the generalizability of the resulting model.

Results for the 2D Pyramid and Cross functions are shown in Figure 2. In the previously used performance measure, the five bracket iterations showed (visually) a higher concentration of individuals in the higher slope regions and therefore suggested superior performance using five brackets.

Our results showed otherwise. As the number of brackets increased, the overall performance *decreased*. This trend was consistent regardless of the dimensionality of the problem. However, the performance relative to the uniform random model was more problem dependent. For the Pyramid problem, bracket counts of one and three were consistently better than random up through 4D, and a bracket count of five was clearly worse. For the 5D Pyramid problem the single bracket count was still better than random. But for the Cross problem the performance did not hold up as the dimensions are increased. Whereas in 2D both one and three brackets were better than random, by 3D only the one-bracket situation was better than random, and by 4D nothing was better to random. The algorithm was not effective in sampling the Michalewicz problem for any number of brackets regardless of dimensionality. Also, in all cases the difference in model performance between random and the algorithm increased with the population size.

These were clearly negative results: a fixed number of brackets did *not* seem to improve the algorithm (unlike visual inspection would suggest) and for the Cross problem, once four dimensions was reached, the algorithm did not outperform random samples.

5 Adaptive Bracketing

The analysis of the fixed bracketing algorithm showed that performance degrades as the bracket count increases and this holds at all points as the population size increases. Even so, it is plausible that the algorithm could perform better if the number of bracket iterations *adaptively* changed based on the current population of individuals. If there is still a lot of exploration that can be performed, additional brackets *should* improve the algorithm performance.

We accomplished adaptive bracketing by incorporating crowding information from the population into the algorithm. If the area in which a child is created is

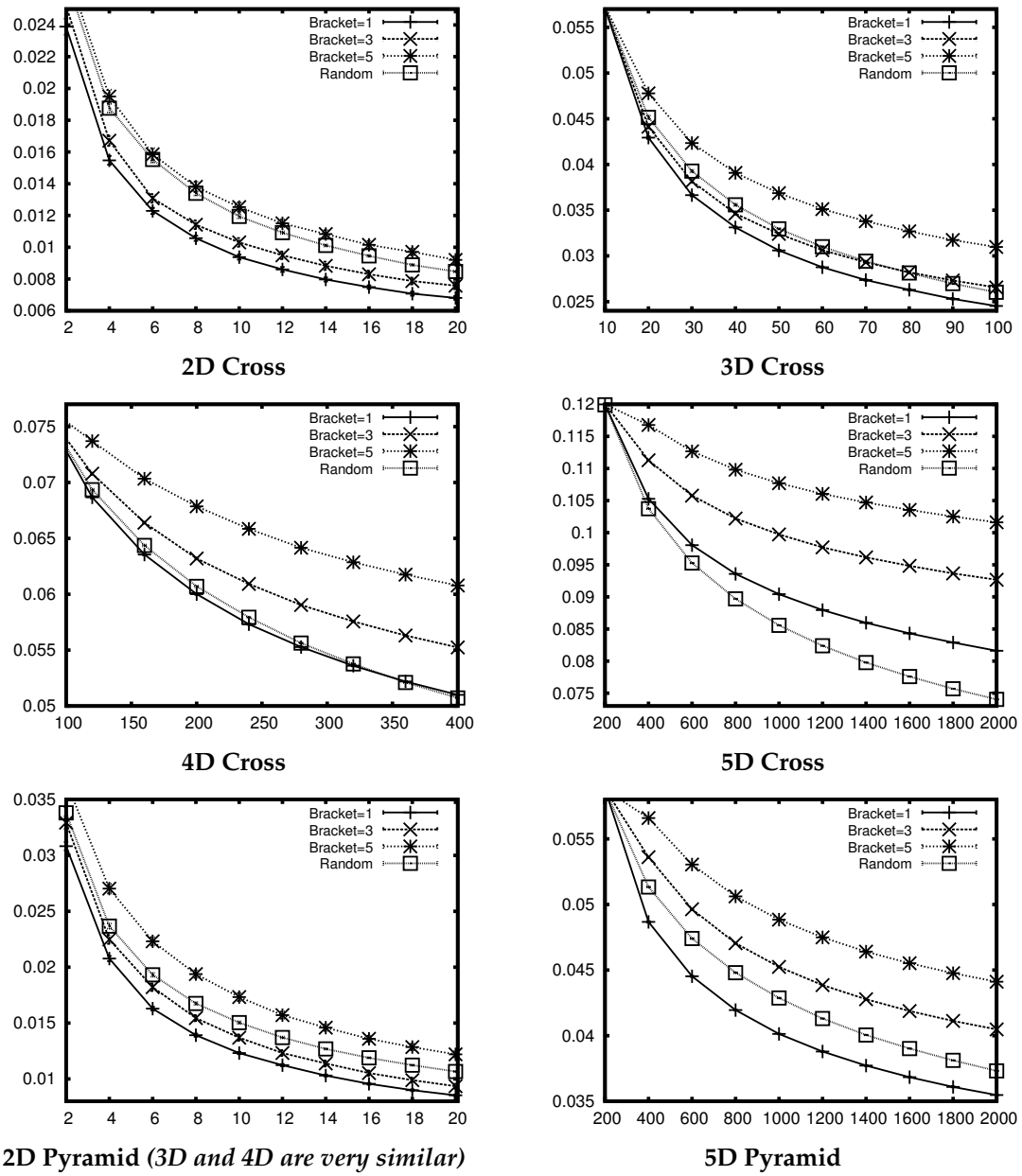


Figure 2: Fixed Bracketing Results. The X Axis is the current population size ($\times$ 1000). The Y Axis is performance (lower is preferred).

crowded, the bracketing loop would be interrupted. Early in the algorithm, the distribution of individuals in parameter space was relatively sparse and a higher number of bracketing iterations can be applied. As the algorithm progressed, the high slope areas would become more densely populated and the number of bracket iterations would decrease, ultimately to one.

5.1 Algorithm Modifications

Implementation of the modifications required the introduction of three additional parameters:

- The *crowding factor* $C_f \in \mathbb{R}|0 < C_f < 1$ is multiplied by the longest diagonal in the parameter space to create a crowding radius. Neighboring individuals that fall within the crowding radius of a reference individual are said to *crowd* that individual.
- If the number of neighboring individuals within the crowding radius exceeds a *crowding limit* $C_\theta \in \mathbb{Z}|C_\theta > 0$, then the reference individual is *crowded*. In all tests, $C_\theta = 10$.
- When a random individual is inserted into the population as parent 1, we don't add it to the population

if it is crowded, unless the maximum number of retries is reached. In all tests the maximum number of random retries was 5.

The algorithm is repeated below with changes in italics.

1. Generate an initial population of uniform random individuals over the parameter space.
2. Choose parent 1 randomly from the population, or according to the exploration probability introduce a new random sample as parent 1. *If the random individual is crowded, try a new random individual up to a maximum number of random retries.*
3. Choose parent 2 from the population using a double tournament.
4. While maximum bracketing count has not been reached:
   (a) Generate a child between parent 1 and parent 2 using linear recombination.
   (b) *"Deactivate" any parent or child that is crowded, so that the individual no longer participates in the selection process.*
   (c) Determine parents for next iteration of the bracketing loop.
   (d) *If either parent for the next iteration of the bracketing loop is inactive, interrupt the loop.*
5. Return to step 2 until the maximum population size has been reached.

The rationale behind deactivating a crowded individual is that once it has become crowded it is unlikely that information can be gained by using that individual as a parent later in the algorithm.

5.2 Analysis

The test method used for the fixed bracketing algorithm was repeated for adaptive bracketing with one difference: the maximum number of brackets was now fixed at five, but the value of the crowding factor was varied. The results are shown in Figure 3. Each plot has five lines defined from top to bottom in the legend as follows:

- A crowding factor value that provided good results (varied with problem).
- The largest crowding factor such that the algorithm would reach the maximum population size. If the crowding factor was too large, the algorithm would deactivate the entire population before the maximum was reached.
- Dynamic adaptive bracketing (described in the next section).
- Results of using the original algorithm using one bracket iteration (Bracket=1).
- Results of using uniform random sampling.

The results for the Cross problem showed crowding to be quite effective in 2D. In 3D, the crowding method still performed marginally better than for Bracket=1, but not until the population size was large. For the 4D problem, the crowding methods were less effective at smaller sizes than Bracket=1 and Random for nearly the entire run. For the 5D problem, the algorithm performance was worse than random, though some crowding methods outperformed Bracket=1 when the population was large.

The results for the Pyramid problem are similar, except that the Bracket=1 method performs better for 3D, 4D, and 5D than other methods until the population size was large. The 4D plot clearly shows that the higher crowding value performs better at smaller population sizes but degrades badly as the population size becomes large.

As for fixed bracketing on the Michalewicz problem, the algorithm could not match the performance of random at any crowding factor. For 2D and 3D the crowding methods outperform Bracket=1 only when the population becomes large. For 4D and 5D the crowding methods do not outperform Bracket=1 at any point during the populations sizes tested.

This meant that crowding *could* help boost bracketing over the ordinary Bracket=1, but usually it required the population sizes to be quite large to be worthwhile.

Furthermore, the larger crowding factor would perform better at lower population sizes (roughly under 10,000) while the smaller crowding factor would perform better at higher population sizes. This observation led us to a dynamic approach described in the next section.

6 Dynamic Adaptive Bracketing

In the analysis of the adaptive crowding algorithm, larger crowding factors performed well at lower population sizes while smaller crowding values performed better at higher population sizes. In addition to the population size, the effectiveness of a crowding factor also depended on the problem and dimensionality. The result was that it required a significant amount of experimentation to determine the most effective crowding factor, which is counterproductive to the goal of minimizing samples in the parameter sweep. If the algorithm were able to start with a high level of crowding and dynamically adjust it down as the algorithm progresses, it should be possible to get performance as good as or better than any fixed crowding value without any experimentation.

6.1 Algorithm Modifications

Implementation of the dynamic crowding introduced five new algorithm parameters. However the algorithm performed relatively well with default values in all but one case. The new algorithm parameters include:

Initial crowding factor: The (relatively high) value of 0.01 worked well for dimensions of 3 or higher. A value of 0.005 was used for 2D.

Crowding evaluation interval: Adjusts the crowding factor each time this many new individuals are added to the population. A value of 500 was used in all cases.

Desired active population size: If the *active* population size exceeded this value, the algorithm would adjust the crowding factor up by the *crowding factor increment* (discussed next). Recall that during adaptive bracketing, crowded individuals were removed from the active population. A good default value, which we used, was the initial population size.

Crowding factor increment: The amount the algorithm increased the crowding factor by if the population was too large. A value of 0.0001 was used in all cases.

Random retry decrement: The amount the algorithm decreased the crowding factor by if the maximum number of random retries (5) was reached. A random retry occurred when the algorithm attempted to add a random individual, but was unable to due to crowding. A value of 0.0005 was used in all cases.

The full algorithm was as follows:

1. Generate an initial population of uniform random individuals over the parameter space.
2. Choose parent 1 randomly from the population, or according to the exploration probability introduce a new random sample as parent 1. If the random individual is crowded, try a new random individual up to a maximum number of retries.
3. Choose parent 2 from the population using a double tournament.
4. While maximum bracketing count has not been reached:
   (a) Generate a child between parent 1 and parent 2 using linear recombination.
   (b) Deactivate any parent or child that is crowded, so that the individual no longer participates in the selection process.
   (c) Determine parents for next iteration of the bracketing loop.
   (d) *If the crowding evaluation interval is reached:*
      i. If the maximum number of random retries was reached during the interval, decrement the crowding factor.
      ii. Otherwise, if the active population size exceeds the target, increment the crowding factor.
   (e) If either parent for the next iteration of the bracketing loop is inactive, interrupt the loop.
5. Return to step 2 until the maximum population size has been reached.

6.2 Analysis

The results of the dynamic adaptive bracketing algorithm, shown in Figure 3, were somewhat inconclusive. In most cases the performance of the dynamic algorithm tracked closely with the best fixed crowding factor. The 4D Pyramid chart shows this most clearly. The dynamic adaptive bracketing plot initially tracks with the larger crowding factor, but eventually tracks to the lower crowding factor when it begins to perform better. As a result, the dynamic algorithm was fairly effective in eliminating the experimentation with crowding factors. However, while the algorithm performed approximately as well as the best adaptive bracketing algorithm for any given problem, it *did not* significantly improve the overall performance at any point.

7 Conclusions

This paper detailed an expansion to [8], which had used a population-based method to attempt to adaptively sample in those regions of the space which were more "interesting", defined as having high slopes. We tackled four major things. First, we recast the search for "interesting" regions of the space into a more formal framework of generalizability to a validation set of samples. While the recasting does not deviate philosophically from the original idea, it allowed us to directly examine the work in [8], and compare it with other techniques, in a quantitative fashion. Second, we examined the original algorithm's bracketing mechanism to see how it performed in terms of the this new measure. Third, we introduced two different algorithms in an attempt to salvage the bracketing procedure given its original unexpected poor performance, and examined their results. Finally, we examined how well these methods would scale with increasing numbers of dimensions.

A New Measure And the Original Algorithm An improved method of performance evaluation was introduced that measures the effectiveness of the parameter sweep as input to a machine learning model. This method scales well to higher dimensions and provided useful insight into the effect of increasing the number of

bracket iterations in the algorithm. Using this measure we discovered that in the original algorithm, in all cases the performance was best when a single child was created between parents and became steadily worse and the number of brackets increased.

Adaptive Bracketing Our second set of experiments tested a potential improvement to the algorithm which adaptively adjusted the number of children created between parents based on the level of crowding in the population. While the algorithm was able to achieve better results in some cases, a high degree of experimentation was required to determine the appropriate level of crowding.

A third set of experiments tested another algorithm modification meant to eliminate the experimentation needed to determine the best level of crowding to adaptively control bracketing. While the these modifications were largely successful in controlling the level of crowding, they did not substantially improve the overall performance.

Scaling In the Pyramid problem, most techniques scaled well with increasing number of dimensions, consistently outperforming Random sampling. But in the simpler Cross problem this was not the case: initially methods would outperform Random sampling, but as the dimensionality increased, Random sampling eventually rose to the top. This may put a damper on the scalability of the technique beyond a moderate number of parameters, as we had imagined might occur.

It should be noted that all forms of the algorithm performed poorly on the Michalewicz problem, regardless of dimensionality or known population size, when compared to the performance of the random model. It is likely that these algorithms are not suitable for such complex problems because they are, well interesting *everywhere*.

References

[1] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] Kenneth De Jong. *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.

[3] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Parallel Problem Solving from Nature (PPSN VI)*, pages 849–858. Springer, 2000.

[4] S. Ghosh and C. R. Rao, editors. *Design and Analysis of Experiments*, volume 13 of *Handbook of Statistics*. Elsevier Science, 1996.

[5] Georges Harik. Finding multimodal solutions using restricted tournament selection. In Larry J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 24–31. Morgan Kaufmann, 1995.

[6] D. Kibler, D. W. Aha, and M. K. Albert. Instance-based prediction of real-valued attributes. *Comput. Intell.*, 5:51–57, May 1989.

[7] T.L. Lew, A.B. Spencer, F. Scarpa, K. Worden, A. Rutherford, and F. Hemez. Identification of response surface models using genetic programming. *Mechanical Systems and Signal Processing*, 20(8):1819 – 1831, 2006.

[8] Sean Luke, Deepankar Sharma, and Gabriel Catalin Balan. Finding interesting things: Population-based adaptive parameter sweeping.

[9] R. Manner, Samir Mahfoud, and Samir W. Mahfoud. Crowding and preselection revisited. In *Parallel Problem Solving From Nature*, pages 27–36. North-Holland, 1992.

[10] Ole J. Mengshoel and David E. Goldberg. The crowding approach to niching in genetic algorithms. *Evolutionary Computation*, 16(3):315–54, 2008.

[11] Ivar Siccama and Maarten Keijzer. Genetic programming as a method to develop powerful predictive models for clinical diagnosis. In *Genetic and Evolutionary Computation Conference (GECCO2005) workshop program*, pages 164–166, Washington, D.C., USA, 25-29 June 2005. ACM Press.

[12] Tony Townsend-Weber and Dennis Kibler. Instance-based prediction of continuous values, 1994.

[13] E.J. Vladislavleva, G.F. Smits, and D. Den Hertog. Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *IEEE Transactions on Evolutionary Computation*, 13(2):333–349, 2009.

[14] G. Gary Wang. Adaptive response surface method using inherited latin hypercube design points. *ASME Transactions, Journal of Mechanical Design*, 125(2):210–220, 2003.

[15] Y. S. Yeun, B. J. Kim, Y. S. Yang, and W. S. Ruy. Polynomial genetic programming for response surface modeling part 2: adaptive approximate models with probabilistic optimization problems. *Structural and Multidisciplinary Optimization*, 29(1):35–49, January 2005.

[16] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In

K. Giannakoglou, D. Tshalis, J. Periaux, K. Papailiou, and T. Fogarty, editors, *Evolutionary Methods for Design, Optimization, and Control*, pages 19–26, 2002.

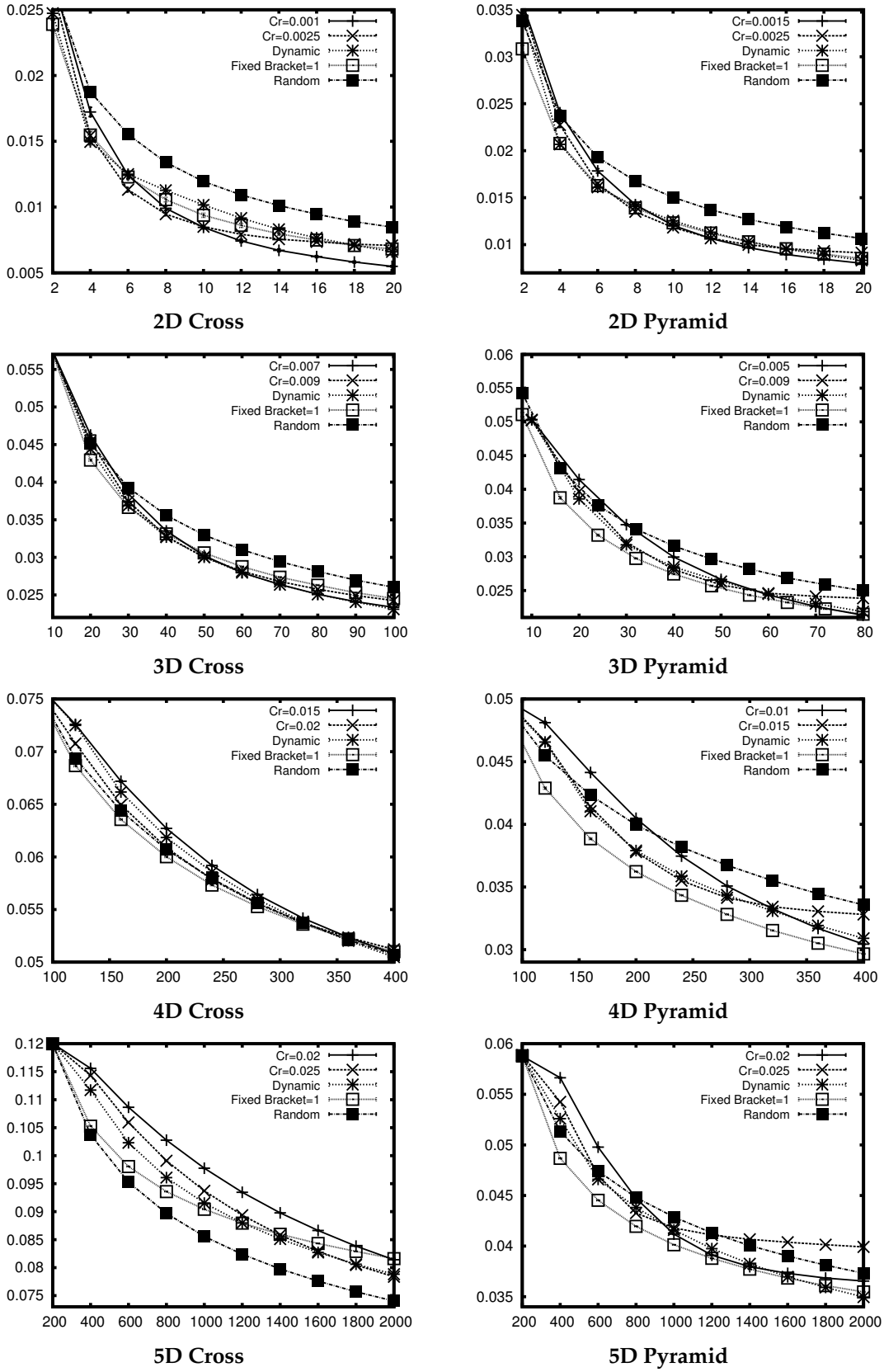


Figure 3: Comparison of Adaptive Bracketing, Dynamic Adaptive Bracketing, a Single Fixed Bracket, and Random points. The X Axis is the current population size ($\times$ 1000). The Y Axis is performance (lower is preferred).